PTO/SB/21 (09-04)
Approved for use through 07/31/2006. OMB 0651-0031
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Total Number of Pages in This Submission: 40

Application Number	10/581,155
Filing Date	31 May 2006
First Named Inventor	Minerva Yeung
Art Unit	
Examiner Name	
Attorney Docket Number	42P18673

ENCLOSURES *(Check all that apply)*

- ☐ Fee Transmittal Form
  - ☐ Fee Attached
- ☐ Amendment/Reply
  - ☐ After Final
  - ☐ Affidavits/declaration(s)
- ☐ Extension of Time Request
- ☐ Express Abandonment Request
- ☐ Information Disclosure Statement
- ☒ Certified Copy of Priority Document(s)
- ☐ Reply to Missing Parts/ Incomplete Application
  - ☐ Reply to Missing Parts under 37 CFR 1.52 or 1.53
- ☐ Drawing(s)
- ☐ Licensing-related Papers
- ☐ Petition
- ☐ Petition to Convert to a Provisional Application
- ☐ Power of Attorney, Revocation Change of Correspondence Address
- ☐ Terminal Disclaimer
- ☐ Request for Refund
- ☐ CD, Number of CD(s) _____
  - ☐ Landscape Table on CD
- ☐ After Allowance Communication to TC
- ☐ Appeal Communication to Board of Appeals and Interferences
- ☐ Appeal Communication to TC **(Appeal Notice, Brief, Reply Brief)**
- ☐ Proprietary Information
- ☐ Status Letter
- ☒ Other Enclosure(s) (please Identify below): Return Receipt Postcard

Remarks

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT

Firm Name	BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN, LLP 12400 Wilshire Boulevard, Seventh Floor, Los Angeles, CA 90025-1030
Signature	
Printed name	**Lester J. Vincent**
Date	June 22, 2006
Reg. No.	**31,460**

CERTIFICATE OF TRANSMISSION/MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service on the date shown below with sufficient postage as first class mail in an envelope addressed to: Mail Stop PCT, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Signature	Theresa Joenks		
Typed or printed name	**Theresa Joenks**	Date	6/22/06

中华人民共和国国家知识产权局
TATE INTELLECTUAL PROPERTY OFFICE
F THE PEOPLE'S REPUBLIC OF CHINA

证 明

CERTIFICATE

本证明之附件是向中国专利局作为受理局提交的下列国际申请副本

S TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY OF THE BELOW NTIFIED INTERNATIONAL APPLICATION THAT WAS FILED WITH THE CHINESE PATENT OFFICE AS RECEIVING OFFICE

请 号: PCT/CN2004/000447

ONAL APPLICATION NUMBER

请 日: 08. MAY 2004(08.05.2004)

LNAL FILING DATE

名 称: FIRMWARE INTERFACE RUNTIME ENVIRONMENT

VENTION PROTECTION FIELD

中华人民共和国国家知识产权局局长

COMMISSIONER OF THE STATE INTELLECTUAL PROPERTY OFFECE OF THE PEOPLE'S REPUBLIC OF CHINA

二零零六年五月十一日

MAY 11, 2006

HOME COPY

P4188/BST

PCT REQUEST

Original (for SUBMISSION)

0	**For receiving Office use only**	
0-1	International Application No.	PCT/CN 2004 / 0 0 0 4 4 7
0-2	International Filing Date	0 8 · MAY 2004 (0 8 · 0 5 · 2 0 0 4)
0-3	Name of receiving Office and "PCT International Application"	RO/CN 中华人民共和国国家知识产权局 PCT International Application
0-4	**Form - PCT/RO/101 PCT Request**	
0-4-1	Prepared Using	PCT-SAFE [EASY mode] Version 3.50 (Build 0002.162)
0-5	**Petition**	
	The undersigned requests that the present international application be processed according to the Patent Cooperation Treaty	
0-6	**Receiving Office (specified by the applicant)**	China Intellectual Property Office (RO/CN)
0-7	**Applicant's or agent's file reference**	P4188/BST
I	**Title of Invention**	FIRMWARE INTERFACE RUNTIME ENVIRONMENT PROTECTION FIELD
II	**Applicant**	
II-1	This person is:	applicant only
II-2	Applicant for	all designated States except US
II-4	Name:	INTEL CORPORATION
II-5	Address:	2200 Mission College Boulevard Santa Clara, California 95052 California United States of America
II-6	State of nationality	US
II-7	State of residence	US
III-1	**Applicant and/or inventor**	
III-1-1	This person is:	applicant and inventor
III-1-2	Applicant for	US only
III-1-4	Name (LAST, First)	CHEN, Lechong
III-1-5	Address:	N31 Apt. 1003 Lane 1555 Kai Xuan Bei Road 200063 Shanghai China
III-1-6	State of nationality	CN
III-1-7	State of residence	CN

P4188/BST

PCT REQUEST

Original (for **SUBMISSION**)

III-2	**Applicant and/or inventor**	
III-2-1	This person is:	applicant and inventor
III-2-2	Applicant for	US only
III-2-4	Name (LAST, First)	XING, Bin
III-2-5	Address:	N12 Apt. 502 Nong 250 Lixi Road, Changning District 200050 Shanghai China
III-2-6	State of nationality	CN
III-2-7	State of residence	CN
III-3	**Applicant and/or inventor**	
III-3-1	This person is:	applicant and inventor
III-3-2	Applicant for	US only
III-3-4	Name (LAST, First)	JIN, Feng
III-3-5	Address:	N13 Apt. 405, 30 Le Shan Road 200030 Shanghai China
III-3-6	State of nationality	CN
III-3-7	State of residence	CN
IV-1	**Agent or common representative; or address for correspondence** The person identified below is hereby/ has been appointed to act on behalf of the applicant(s) before the competent International Authorities as:	agent
IV-1-1	Name:	KANGXIN & PARTNERS
IV-1-2	Address:	Suite 402 Block B, Tongtai Mansion, Jinrongdajie, Xicheng District, 100032 Beijing China
IV-1-3	Telephone No.	0086-010 88087204
IV-1-4	Facsimile No.	0086-010-88087200
IV-1-5	e-mail	gpliu@vip.sina.com
IV-1-6	Agent's registration No.	11240
V	**DESIGNATIONS**	
V-1	**The filing of this request constitutes under Rule 4.9(a), the designation of all Contracting States bound by the PCT on the international filing date, for the grant of every kind of protection available and, where applicable, for the grant of both regional and national patents.**	
VI-1	**Priority Claim**	NONE

P4188/BST

PCT REQUEST

Original (for **SUBMISSION**)

VII-1	**International Searching Authority Chosen**	China Intellectual Property Office (ISA/CN)	
VIII	**Declarations**	Number of declarations	
VIII-1	Declaration as to the identity of the inventor	-	
VIII-2	Declaration as to the applicant's entitlement, as at the international filing date, to apply for and be granted a patent	-	
VIII-3	Declaration as to the applicant's entitlement, as at the international filing date, to claim the priority of the earlier application	-	
VIII-4	Declaration of inventorship (only for the purposes of the designation of the United States of America)	-	
VIII-5	Declaration as to non-prejudicial disclosures or exceptions to lack of novelty	-	
IX	**Check list**	number of sheets	electronic file(s) attached
IX-1	Request (including declaration sheets)	4	✓
IX-2	Description	18	-
IX-3	Claims	7	-
IX-4	Abstract	1	✓
IX-5	Drawings	7	-
IX-7	TOTAL	37	
	Accompanying Items	paper document(s) attached	electronic file(s) attached
IX-8	Fee calculation sheet	✓	-
IX-17	PCT-SAFE physical media	-	✓
IX-19	**Figure of the drawings which should accompany the abstract**	2	
IX-20	**Language of filing of the international application**	English	
X-1	**Signature of applicant, agent or common representative**		
X-1-1	Name:	KANGXIN & PARTNERS	
X-1-2	Name of signatory	Guoping LIU	
X-1-3	Capacity		

P4188/BST

PCT REQUEST

Original (for **SUBMISSION**)

FOR RECEIVING OFFICE USE ONLY

10-1	Date of actual receipt of the purported international application	0 8 · MAY 2004 (0 8 · 0 5 · 2 0 0 4)
10-2	Drawings:	
10-2-1	Received	
10-2-2	Not received	
10-3	Corrected date of actual receipt due to later but timely received papers or drawings completing the purported international application	
10-4	Date of timely receipt of the required corrections under PCT Article 11(2)	
10-5	International Searching Authority	ISA/CN
10-6	Transmittal of search copy delayed until search fee is paid	

FOR INTERNATIONAL BUREAU USE ONLY

11-1	Date of receipt of the record copy by the International Bureau	

P4188/BST

PCT (ANNEX - FEE CALCULATION SHEET)

Original (for **SUBMISSION**)

(This sheet is not part of and does not count as a sheet of the international application)

0	**For receiving Office use only**	
0-1	International Application No.	PCT/CN2004 / 000447
0-2	Date stamp of the receiving Office	08 · MAY 2004 (08 · 05 · 2004)

0-4	**Form PCT/RO/101 (Annex)** **PCT Fee Calculation Sheet**				
0-4-1	Prepared Using		PCT-SAFE [EASY mode] Version 3.50 (Build 0002.162)		
0-9	**Applicant's or agent's file reference**		P4188/BST		
2	**Applicant**		INTEL CORPORATION		
12	**Calculation of prescribed fees**		fee amount/muliplier	Total amounts (CNY)	Total amounts (EQF)
12-1	Transmittal fee	T	⇨	500	
12-2-1	Search fee	S	⇨	1500	
12-2-2	International search to be carried out by		CN		
12-3	International filing fee (first 30 sheets)	i1	1400 EQF		
12-4	Remaining sheets		7		
12-5	Additional amount	(X)	15 EQF		
12-6	Total additional amount	i2	105 EQF		
12-7	i1 + i2 =	i	1505 EQF		
12-12	EASY Filing reduction	R	EQF-100		
12-13	Total International filing fee (i-R)	I	⇨		1405
12-14	Fee for priority document Number of priority documents requested		0		
12-15	Fee per document	(X)	150 CNY		
12-16	Total priority document fee:	P	⇨		
12-17	**TOTAL FEES PAYABLE (T+S+I+P)**		⇨	2000	1405
12-19	**Mode of payment**		authorization to charge deposit account		

FIRMWARE INTERFACE RUNTIME ENVIRONMENT PROTECTION FIELD

[0001] Embodiments of the invention relate to firmware interfaces of a platform; and more specifically, to firmware interface runtime environment protection.

BACKGROUND

[0002] For EFI (extensible firmware interface) based BIOS (basic input/output system), there are some critical and valuable data structures and code that need to persist in a system memory at runtime, such as the S3 boot script table, EFI system table, EFI runtime services, etc. S3 resume functionality of EFI based BIOS is built upon the S3 boot script table. EFI runtime services will be used by an Operating System (OS). Without a protection mechanism, they are vulnerable to attack by the virus running at runtime. As such, the virus that has that EFI specific knowledge can take over control of the system by replacing the EFI S3 boot script or other runtime data, with its own rogue version. Executing this rogue routine will cause severe consequences, including critical end user data leakage.

[0003] Figure 1 is a block diagram illustrating a typical configuration initialization of a platform (e.g., a data processing system). Referring to Figure 1, in an EFI based environment, the platform is initialized in a phased fashion in the normal boot path 101. There are two phases for the platform initialization: Pre-EFI Initialization (PEI) 102, followed by Driver Execution Environment (DXE) 103. During the PEI phase 102, it initializes the minimum system resources to enable the DXE phase 103. During the DXE phase 103, numerous DXE drivers are executed collectively to initialize the platform into the final pre-boot state OS load 104. The majority of platform initialization is accomplished in the DXE phase 103 as it has much richer resources than the PEI phase 102.

[0004] In contrast, in S3 resume boot path 108, in order to achieve high-performance S3 restoration, a mechanism called EFI Boot Script is introduced to avoid executing the DXE phase 110 which is too complicated and time-consuming against the very strict requirement of S3 resume time. The process of the platform initialization can be viewed as a sequence of

operations including accessing the I/O, memory, and PCI configuration space, and executing specific microprocessor instructions.

[0005] All of the above operations can be represented in EFI boot scripts. As such, the platform initialization can be performed by executing a sequence of EFI boot scripts of script table 107. During a normal boot path 101, the DXE drivers record their platform initialization operations as some boot scripts. Before booting the OS (block 104), all of these boot scripts are organized as a boot script table 107 and the boot script table 107 is copied into an Advanced Configuration and Power Interface (ACPI) Non-Volatile Storage (NVS) memory region 105 which will not be perturbed by the OS at runtime.

[0006] When the system wakes up and runs the S3 resume boot path 108, PEI module 109 of a boot script engine is able to execute all of the boot scripts in the boot script table 107 to restore (block 110) the configuration done in the previous DXE phase (e.g., block 103), instead of executing the DXE phase. This mechanism can expedite S3 resume. However, because this mechanism S3 resume highly relies on the boot script table 107 which is stored in the system memory and persists through runtime, the boot script table 107 is vulnerable to attack by viruses during runtime. One EFI boot script named dispatch boot script to perform the processor initialization. Dispatch boot script just records the entry point of a piece of arbitrary code. The action taken on execution of the dispatch boot script is just jumping to the entry point as used to execute the code (e.g., code 112). Code 112 will also be stored into ACPI NVS and persist through runtime. This code may be modified by malevolent code running at the runtime. Thus, by attacking the boot script table 107 and the code to be executed 112 during the execution of the boot scripts, viruses running in runtime environment can easily change S3 resume behavior, thereby taking over the control of the system.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The invention may best be understood by referring to the following description and accompanying drawings that are used to illustrate embodiments of the invention. In the drawings:

[0008] Figure 1 is a block diagram illustrating a typical configuration initialization of a platform.

[0009] Figure 2 is a block diagram illustrating an example of an EFI initialization of a platform according to one embodiment.

[0010] Figure 3 is a block diagram illustrating an example of an initialization mechanism of a platform according to one embodiment.

[0011] Figure 4 is a flow diagram illustrating a process example for initializing a platform according to one embodiment.

[0012] Figure 5 is a flow diagram illustrating a process example for initializing a platform according to one embodiment.

[0013] Figure 6 is a block diagram illustrating an EFI architecture example that may be used with an embodiment of the invention.

[0014] Figure 7 is a block diagram illustrating an example of a data processing system according to one embodiment of the present invention.

DETAILED DESCRIPTION

[0015] Method and apparatus for protecting a firmware runtime environment are described herein. In one embodiment, a variety of security techniques, such as, for example, digital signature or HMAC (HMAC: Keyed-Hashing for Message Authentication, RFC-2104), depending on what kind of secure store is available, are used to ensure the integrity of the critical runtime data structures and code for EFI based BIOS. In one embodiment, a secure store is implemented on the platform to protect keys, which are used to generate a signature or a HMAC, such that an attacker cannot forge a signature or a HMAC at runtime. In one embodiment, such a secure store may be implemented using a variety of techniques, such as, for example, SMRAM (system management RAM), secure flash, and/or TPM (trusted platform module), etc. In one embodiment, a signature (or HMAC) can be generated for both the boot script table and the code to be dispatched in a normal boot path, and then verified in an S3 resume boot path. As a result, any modifications to either the boot script table or the code to be dispatched by an attacker (or virus) can be detected.

[0016] In the following description, numerous specific details are set forth. However, it is understood that embodiments of the invention may be practiced without these specific details. In other instances, well-known circuits, structures and techniques have not been shown in detail in order not to obscure the understanding of this description.

[0017] Some portions of the detailed descriptions which follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0018] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar data processing device, that manipulates and transforms data represented as physical (e.g. electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0019] Embodiments of the present invention also relate to apparatuses for performing the operations described herein. An apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs) such as Dynamic RAM (DRAM), erasable programmable ROMs (EPROMs), electrically erasable programmable ROMs (EEPROMs), magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each of the above storage components is coupled to a computer system bus.

[0020] The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the methods. The structure for a variety of these systems will appear from the description below. In addition, embodiments of the present invention are not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the embodiments of the invention as described herein.

[0021] A machine-readable medium includes any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory ("ROM"); random access memory ("RAM"); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.); etc.

[0022] Figure 2 is a block diagram illustrating an example of an EFI initialization of a platform according to one embodiment. A signature (also referred to as a signed hash) is generated for the boot script table and stored along with the table. A signature (e.g., signed hash) is also generated for each piece of code to be dispatched, and is stored along with the corresponding dispatch boot item in the table. When the platform resumes from S3, a PEI boot script engine invokes a verifier to verify the integrity of the boot script table before executing any boot scripts and verify the code to be dispatched before executing the code to be dispatched. If the verification fails at any time, the system will be reset to prevent it from executing any rogue code. As a result, the S3 resume boot path is secured for EFI based BIOS.

[0023] Referring to Figure 2, similar to the configuration shown in Figure 1, during a normal boot path 201, there is a PEI phase 202 and a DXE phase 203, prior to the OS load phase 204. During a resume path 208, there is also a PEI phase 209 and a DXE phase 210 before handing the control over to the waking vector phase 211. In one embodiment, during the normal boot path 201, a key is generated to sign the boot script table 207. The key generated may be a key pair, such as, for example, a RSA (Rivest, Shamir, and Adleman) key pair or a PGP (Pretty Good Privacy) key pair. Alternatively, the key may be a symmetrical key, such as, for example, an HMAC key. In one embodiment, the boot script table is hashed and signed using an asymmetric key.

[0024] In one embodiment, the keys generated may be stored in a secure store that can only be read or has no access at all after the boot time. In one embodiment, the secure store may be implemented as a store that satisfies one or more predetermined security

requirements or policies. That is, during the boot time, the secure store may be read and written, during which the keys are generated and stored in the secure store. After the keys are stored in the secure store, the secure store may be locked to have either only read access or no access at all. The secure store can only be unlocked when the platform is reset. As a result, the keys are secured to deter virus attacks.

[0025] In one embodiment, the boot script table (also referred to as an initialization table) 207 includes a header 212 and one or more boot scripts (also referred to as initialization segments) 213-215. Some of the boot scripts may contain code that can be dispatched and executed during boot time. These boot scripts may be referred to as dispatch boot scripts that only contain an entry point (e.g., a pointer) to a piece of code to be dispatched. For example, boot script 214 includes code 206 to be dispatched at boot time. In one embodiment, some of the boot scripts are signed using a key generated above. In a particular embodiment, a boot script having code to be dispatched may require to be signed with a key. For example, boot script 214 having code 206 to be dispatched may be signed using a key and the resulting signature may be stored as signed hash 217 associated with boot script 214. In addition, the boot script table 207 may be signed using another key and the resulting signature may be stored as signed hash 216. It will be appreciated that the keys for signing the code to be dispatched and the key for signing the boot script table may or may not be the same key.

[0026] During the resume path 208, according to one embodiment, a boot script engine (e.g., boot script engine 301 of Figure 3) handling the boot scripts may retrieve the keys from the secure store (e.g., secure store 303 of Figure 3) and use the keys to verify the boot script table and the boot scripts whether the data integrity is still valid. For example, during the resume process, the boot script engine retrieves a key from the secure store to verify the signed hash 216 for the boot script table 207. In addition, the boot script engine may further retrieve another key or use the same key to verify the signed hash for the code associated with each of the dispatch boot scripts, such as signed hash 217 of boot script 214.

[0027] In one embodiment, the verification of the boot scripts may only be performed if the respective boot script contains code to be dispatched. Once the verification is performed successfully, according to one embodiment, the code associated with the boot script may be

dispatched and executed. As a result, even if an attacker replaced the code to be dispatched, the verification would not be performed successfully, and the code would not be dispatched and executed. If the respective boot script does not contain code to be dispatched, according to one embodiment, the boot script will be executed without further verification.

[0028] Figure 3 is a block diagram illustrating an example of an initialization mechanism of a platform according to one embodiment. The initialization mechanism example 300 may be implemented within a firmware of a platform (e.g., a data processing system or computer). For example, the initialization mechanism 300 may be implemented as a part of an EFI of a platform. In one embodiment, the initialization mechanism 300 includes, but is not limited to, an initialization engine to perform operations of an initialization table for initializing a platform, a secure store to store one or more keys for signing at least a portion of the initialization table, and a verifier coupled to the initialization engine and the secure store to verify at least a portion of the initialization table using at least one of the one or more keys during an initialization of the platform.

[0029] Referring to Figure 3, the initialization mechanism 300 includes a boot script engine (also referred to as an initialization engine) 301, a verifier 302, a secure store 303, and a boot script table (also referred to as an initialization table) 305. In one embodiment, the boot script engine may be responsible for executing one or more boot scripts of the boot script table to perform the initialization operations. The verifier 302 is responsible for verifying at least a portion of the boot script table during an initialization phase, such as, for example, a boot resume phase, of a platform.

[0030] In one embodiment, the boot script table 305 includes, but is not limited to, a table header 306, one or more boot scripts (also referred to as initialization segments) 307-309. In one embodiment, at least a portion of the boot script table 305 may be signed (e.g., encrypted and/or hashed) using a key, which may be a part of keys 304 stored in the secure store 303. The signature or signatures of the boot script table may be stored as signed hash 311 associated with the boot script table 305. In one embodiment, one or more boor scripts may further be signed by a key, which may be a part of keys 304 stored in the secure store 303.

[0031] Some of the boot scripts, such as, boot script 308, of the boot script table 305 may contain code (e.g., code 206 of Figure 2) that can be dispatched and executed. The boot script 308 may be a dispatch script containing only a reference to a dispatchable code. In one embodiment, only the boot script having the code to be dispatched may be signed with a key, either the same key or a different key. As a result, even if an attacker replaced the code to be dispatched, the replaced code would not be verified successfully by the verifier 302 and thus, the replaced code would be dispatched and/or executed. The above described security processes may be performed using a variety of security techniques. For example, the keys 304 may be key pairs, such as, for example, RSA or PGP key pairs. Alternatively, the keys 304 may be symmetrical keys.

[0032] The secure store 303 may be implemented using a variety of techniques, such as, for example, SMRAM (system management RAM), secure flash, and/or TPM (trusted platform module), etc. In one embodiment, the secure store 303 may be read-write on the initial power-on. The secure store 303 may be able to be locked so that it becomes read-only. The secure store may be unlocked only when the platform is reset.

[0033] Referring to Figure 3, in one embodiment as an example, an RSA key pair is generated randomly in the normal boot path (e.g., normal boot path 201 of Figure 2). Then the firmware (e.g., boot script engine 301) uses the generated private key of the key pair to sign the boot script table 305 and the code to be dispatched of boot script 308. Note that the boot script table and the code to be dispatched may be signed using the same key, or alternatively, using different keys. In a particular embodiment, the data structure of the code to be dispatched is hashed and signed using the private key. However, it is not necessary to follow those digital signature format specifications defined in PKCS #7 or any other standards. Other security techniques may be utilized.

[0034] Thereafter, according to one embodiment, the public key is stored in the secure store 303 as a part of keys 304 and the secure store is locked. Once the secure store is locked, the private key is destroyed before passing the control to the OS loader (e.g., OS loader 204 of Figure 2). Since the public key is locked in the secure store and its corresponding private

key is destroyed, an attacker cannot tamper with the public key, nor can the attacker tamper with the boot script table 305 and forge a signature (e.g., signed hash 311). Note that it is not necessary to use an RSA algorithm to generate signatures. Any other asymmetric signing algorithm could be used.

[0035] According to another embodiment, the secure store 303 may be read-write on the initial power-on. The secure store 303 may be able to be locked so that no access from outside of the secure store is available while it is locked. The secure store may be unlocked only when the platform is reset.

[0036] In this embodiment, a symmetric key is generated randomly during the normal boot path (e.g., normal boot path 201 of Figure 2). The firmware uses the symmetric key to calculate a HMAC for the boot script table 305 (or to encrypt the data structure if the privacy is desired) and the code to be dispatched (e.g., code 206). Thereafter, the symmetric key is stored as a part of keys 304 in the secure store 303 before passing control to the OS loader (e.g., OS loader 204 of Figure 2). As a result, an attacker cannot forge an HMAC even if he has tampered with either the boot script table 305 or the code to be dispatched since the key is locked in the secure store 303 and thereby cannot be accessed. In one embodiment, the secure store that allows an execution of code therein, such as SM Ram, may also expose an interface for HMAC verification at runtime.

[0037] In this embodiment, an assumption is made, that the code that verifies the signatures (e.g., the verifier 302) is intact. In one embodiment, the firmware hub can be locked so that the flash can be treated as a read-only storage. When the platform resumes from S3, the code in the boot block of the flash will be executed at the very beginning. The boot block is responsible to make sure that the verifier 302 will be intact and behave as expected. In one embodiment, the verifier 302 may be loaded first from the flash, which can be treated read-only on most platforms.

[0038] It will be appreciated that the signatures are not necessary to be stored next to the corresponding boot script. In fact, the signature could be stored anywhere, as long as the verifier 302 can find them when needed. In addition, the processes are similar if HMAC is used instead of RSA key pairs, except that HMAC requires the secure store 303 to be

inaccessible by the code outside the secure store when the secure store is locked, in which case, the verifier 302 may also be located in the secure store and expose interfaces for HMAC verification. Other configurations and/or implementations may be utilized.

[0039] Figure 4 is a flow diagram illustrating a process example for initializing a platform according to one embodiment. Process example 400 may be performed by a processing logic that may comprise hardware (circuitry, dedicated logic, etc.), software (such as is run on a dedicated machine), or a combination of both. For example, the process example 400 may be performed during a normal boot path, such as, normal boot path 201 of Figure 2. In one embodiment, process example 400 includes, but not limited to, generating a first key to sign an initialization table (e.g., boot script table) of a firmware in a platform, the initialization table being used to initialize the platform, signing the initialization table using the first key, storing the first key in a secure store that satisfies one or more security requirements or policies, and locking the secure store after the first key is stored in the secure store.

[0040] Referring to Figure 4, at block 401, a first key is generated to sign (e.g., encrypt and/or hash) a boot script table, such as, boot script table 305 of Figure 3, and one or more second keys are generated to sign (e.g., encrypt and/or hash) the dispatchable code of one or more boot scripts (e.g., boot script 308 of Figure 3). In one embodiment, the second keys are generated only for the code to be dispatched. The one or more second keys may be generated specifically for each boot script. Alternatively, the same second key may be used to sign all of the boot scripts having code to be dispatched. Furthermore, the first and second keys may be the same key.

[0041] At block 402, the dispatchable code of one or more boot scripts are signed with the one or more second keys and at block 403, the boot script table is signed with the first key. At block 404, the first and second keys are stored in a secure store, such as secure store 303 of Figure 3 and thereafter, at block 405, the control may be passed over to the OS loader.

[0042] According to one embodiment, the first and second keys may be generated as key pairs, such as, RSA and/or PGP key pairs. Alternatively, the first and second keys may be generated as symmetric keys. When the key pairs are used, in one embodiment, the secure

store may be accessed as read/write during initial power-on processes, locked as read-only after the first and second keys are stored therein, and unlocked only when the platform is reset. When the symmetric keys are used, according to another embodiment, the secure store may be accessed as read/write during initial power-on processes, locked without any access from outside of the secure store after the first and second keys are stored therein, and unlock only when the platform is reset. Other configurations may exist.

[0043] Figure 5 is a flow diagram illustrating a process example for initializing a platform according to one embodiment. Process example 500 may be performed by a processing logic that may comprise hardware (circuitry, dedicated logic, etc.), software (such as is run on a dedicated machine), or a combination of both. For example, the process example 500 may be performed during a resume boot path, such as, resume boot path 208 of Figure 2. In one embodiment, process example 500 includes, but not is limited to, retrieving a first key from a secure store within a platform, the firmware including an initialization table for initializing the platform, and verifying the initialization table using the first key retrieved from the secure store during an initialization of the platform.

[0044] Referring to Figure 5, during an initialization of a platform, such as a resume boot process, at block 501, a first key is retrieved from a secure store, such as secure store 303 of Figure 3. The boot script table, such as boot script table 305 is verified using the first key. The boot script table is signed (e.g., encrypted and/or hashed) previously using the first key during a previous initialization of the platform, such as a normal boot process. If the verification is performed unsuccessfully, at block 508, the platform is reset.

[0045] If the verification is performed successfully, at block 502, it is determined whether a boot script of the boot script table contains code that can be dispatched. If the respective boot script does not contain the code to be dispatched (e.g., boot script 307 of Figure 3), at block 503, the boot script is executed without further verification and a next boot script is processed.

[0046] If the respective boot script contains the code to be dispatched (e.g., boot script 308), at block 504, a second key is retrieved from the secure store and the code to be dispatched is verified using the second key. The code of the respective boot script is signed

(e.g., encrypted and/or hashed) previously using the second key during a previous initialization of the platform, such as a normal boot process. If the verification of the boot script is performed unsuccessfully, at block 508, the platform is reset.

[0047] If the verification of the boot script is performed successfully, at block 505, the code to be dispatched corresponding to the boot script is executed. The above processes are repeated until all of the boot scripts of the boot script table have been processed (block 506). Thereafter, at block 507, the control is transferred to the OS waking vector. Other operations may also be performed.

[0048] According to one embodiment, the first and second keys may be generated as key pairs, such as, RSA and/or PGP key pairs. Alternatively, the first and second keys may be generated as symmetric keys. When the key pairs are used, in one embodiment, the secure store may be accessed as read/write during initial power-on processes, locked as read-only after the first and second keys are stored therein, and unlock only when the platform is reset. When the symmetric keys are used, according to another embodiment, the secure store may be accessed as read/write during initial power-on processes, locked without any access from outside of the secure store after the first and second keys are stored therein, and unlocked only when the platform is reset. Other configurations may exist.

[0049] Figure 6 is a block diagram illustrating an EFI architecture example that may be used with an embodiment of the invention. Referring to Figure 6, in one embodiment, the architecture example 600 includes, but is not limited to, an operating system (OS) 601, an EFI OS loader 602, EFI boot services 603, EFI runtime services 604, platform hardware/firmware 605, and interfaces for other specifications or standards 606.

[0050] OS 601 may be an operating system from a variety of vendors, such as, for example, a Windows operating system from Microsoft or a Mac OS from Apple Computer. Alternatively, the OS 601 may be UNIX or Linux operating system. Other operating systems, such as, for example, embedded operating systems or real-time operating systems may be utilized. OS loader 602 is responsible for loading OS 601.

[0051] EFI boot services 603 provide interfaces for devices and system functionality that can be used during boot time. Device access is abstracted through "handles" and

"protocols." This facilitates reuse of investments out of the specification without burdening the consumer accessing the device. EFI runtime services 604 are used to ensure appropriate abstraction of base platform hardware resources that may be needed by the OS during the normal operations.

[0052] In one embodiment, platform firmware/hardware 605 includes, but is not limited to, an EFI system partition that may include an EFI OS loader. The system partition defines a partition and file system that are designed to allow safe sharing between multiple vendors, and for different purposes. The ability to include a separate sharable system partition presents an opportunity to increase platform value-add without significantly growing the need for non-volatile platform memory.

[0053] The platform firmware is able to retrieve the OS loader image from the EFI system partition 607. The specification provides for a variety of mass storage device types including disk, CD-ROM, and DVD, as well as remote boot via a network. Through the extensible protocol interfaces, it is possible to envision other boot media types being added, although these may require OS loaded modifications if they require use of specific protocols other than those standardized.

[0054] Once started, the OS loader 602 continues to boot the complete operating system 601. To do so, it may use the EFI boot services 603 to survey, comprehend, and initialize the various platform components and the OS software that manages them. EFI runtime services 604 may also be available to the OS loader 602 during the boot phase.

[0055] In addition, the platform firmware/hardware 605 includes a secure store 608, where one or more keys 609 may be stored. The keys 609 may include the keys to sign a boot script table (e.g., boot script table 305 of Figure 3) and dispatchable code of one or more boot scripts (e.g., boot scripts 307-309 of Figure 3). The keys 609 may be generated during a normal boot path (e.g., normal boot path 201 of Figure 2) and used to verify the boot script table and one or more boot scripts during a resume boot path (e.g., resume boot path 208 of Figure 2). The secure store 608 may be implemented using a variety of techniques, such as, for example, SMRAM (system management RAM), secure flash, and/or TPM (trusted platform module), etc. Note that the secure store 608 may not necessarily within the

platform firmware 605. It will be appreciated that the secure store 608 may be implemented anywhere within the platform as long as the secure store 608 can be accessed by the firmware and satisfies a set of security policies.

[0056] According to one embodiment, the keys 609 may be generated as key pairs, such as, RSA and/or PGP key pairs. Alternatively, the keys 609 may be generated as symmetric keys. When the key pairs are used, in one embodiment, the secure store 608 may be accessed as read/write during initial power-on processes, locked as read-only after the keys 608 are stored therein, and unlocked only when the platform is reset. When the symmetric keys are used, according to another embodiment, the secure store 608 may be accessed as read/write during initial power-on processes, locked without any access from outside of the secure store after the keys 609 are stored therein, and unlocked only when the platform is reset. Other configurations may exist.

[0057] Figure 7 is a block diagram illustrating an example of a data processing system according to one embodiment of the present invention. The exemplary system 700 may be used to perform the process examples described above to protect runtime environments. Note that while Figure 7 illustrates various components of a computer system, it is not intended to represent any particular architecture or manner of interconnecting the components, as such details are not germane to the present invention. It will also be appreciated that network computers, handheld computers, cell phones, and other data processing systems, which have fewer components or perhaps more components, may also be used with the present invention. The computer system of Figure 7 may, for example, be an Apple Macintosh computer or an IBM compatible PC.

[0058] Referring to Figure 7, the computer system 700 includes, but not limited to, a processor 702 that processes data signals. The processor 702 may be a complex instruction set computer (CISC) microprocessor, a reduced instruction set computing (RISC) microprocessor, a very long instruction word (VLIW) microprocessor, a processor implementing a combination of instruction sets, or other processor device, such as a digital signal processor, for example. Figure 7 shows an example of an embodiment of the invention implemented as a single processor system 700. However, it is understood that embodiments

of the present invention may alternatively be implemented as systems having multiple processors. Processor 700 may be coupled to a processor bus 710 that transmits data signals between processor 702 and other components in the system 700.

[0059] In addition, system 700 includes a memory 716. Memory 716 may be a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, or other memory device. Memory 716 may also contain additional software and/or data not shown. A cache memory 704 may reside inside or outside the processor 702 that stores data signals stored in memory 716. Cache memory 704 in this embodiment speeds up memory accesses by the processor by taking advantage of its locality of access.

[0060] Further, a bridge/memory controller 714 may be coupled to the processor bus 710 and memory 716. The bridge/memory controller 714 directs data signals between processor 702, memory 716, and other components in the system 700 and bridges the data signals between processor bus 710, memory 716, and a first input/output (I/O) bus 720. In some embodiments, the bridge/memory controller provides a graphics port for coupling to a graphics controller 712. In this embodiment, graphics controller 712 interfaces to a display device for displaying images rendered or otherwise processed by the graphics controller 712 to a user. The display device may include a television set, a computer monitor, a flat panel display, or other suitable display devices.

[0061] First I/O bus 720 may include a single bus or a combination of multiple buses. First I/O bus 720 provides communication links between components in system 700. A network controller 722 may be coupled to the first I/O bus 720. The network controller links system 700 to a network that may include a plurality of processing system and supports communication among various systems. The network of processing systems may include a local area network (LAN), a wide area network (WAN), the Internet, or other network.

[0062] In some embodiments, a display device controller 724 may be coupled to the first I/O bus 720. The display device controller 724 allows coupling of a display device to system 700 and acts as an interface between a display device and the system. The display device may comprise a television set, a computer monitor, a flat panel display, or other suitable display device. The display device receives data signals from processor 702 through display

device controller 724 and displays information contained in the data signals to a user of system 700.

[0063] A second I/O bus 730 may comprise a single bus or a combination of multiple buses. The second I/O bus 730 provides communication links between components in system 700. A data storage device 732 may be coupled to second I/O bus 730. The data storage device 732 may include a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage devices. Data storage device 732 may include one or a plurality of the described data storage devices.

[0064] A user input interface 734 may be coupled to the second I/O bus 730, such as, for example, a keyboard or a pointing device (e.g., a mouse). The user input interface 734 may include a keyboard controller or other keyboard interface device. The user input interface 734 may include a dedicated device or may reside in another device such as a bus controller or other controller device. The user input interface 734 allows coupling of a user input device (e.g., a keyboard, a mouse, joystick, or trackball, etc.) to system 700 and transmits data signals from a user input device to system 700.

[0065] One or more I/O controllers 738 may be used to connect one or more I/O devices to the exemplary system 700. For example, the I/O controller 738 may include a USB (universal serial bus) adapter for controlling USB peripherals or alternatively, an IEEE 1394 (also referred to as Firewire) bus controller for controlling IEEE 1394 compatible devices.

[0066] Furthermore, the elements of system 700 perform their conventional functions well-known in the art. In particular, data storage device 732 may be used to provide long-term storage for the executable instructions and data structures for embodiments of methods of dynamic loop aggregation in accordance with embodiments of the present invention, whereas memory 716 is used to store on a shorter term basis the executable instructions of embodiments of the methods of dynamic loop aggregation in accordance with embodiments of the present invention during execution by processor 702.

[0067] Although the above example describes the distribution of computer code via a data storage device, program code may be distributed by way of other computer readable mediums. For instance, a computer program may be distributed through a computer readable

medium such as a floppy disk, a CD ROM, a carrier wave, a network, or even a transmission over the Internet. Software code compilers often use optimizations during the code compilation process in an attempt to generate faster and better code.

[0068] Thus, method and apparatus for protecting a firmware runtime environment have been described herein. In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will be evident that various modifications may be made thereto without departing from the broader spirit and scope of the invention as set forth in the following claims. The specification and drawings are, accordingly, to be regarded in an illustrative sense rather than a restrictive sense.

CLAIMS

What is claimed is:

1. A method, comprising:

retrieving a first key from a secure store associated with a firmware within a platform, the firmware including an initialization table for initializing the platform; and

verifying the initialization table using the first key retrieved from the secure store during an initialization of the platform.

2. The method of claim 1, wherein the initialization table comprises one or more initialization segments that are individually executable, and wherein the method further comprises:

retrieving a second key from the secure store; and

verifying at least one initialization segment using the second key retrieved from the secure store.

3. The method of claim 2, wherein the at least one initialization segment is signed using the second key prior to being stored in the firmware.

4. The method of claim 2, further comprising examining the at least one initialization segment to determine whether the at least one initialization segment includes code to be dispatched, wherein the verification is performed only if the at least one initialization segment includes the code to be dispatched.

5. The method of claim 4, further comprising:

determining whether the verification is performed successfully; and

executing the code dispatched from the at least one initialization segment if the verification is performed successfully.

6. The method of claim 5, further comprising resetting the platform if the verification is performed unsuccessfully.

7. The method of claim 2, further comprising executing the at least one initialization segment without performing the verification, if the at least one initialization segment does not include the code to be dispatched.

8. The method of claim 1, wherein the initialization of the platform is performed during a resume process of the platform, and wherein the first and second keys are generated, and the initialization table and the at least one initialization segment are signed during a boot process of the platform.

9. A machine-readable medium having executable code to cause a machine to perform a method, the method comprising:

retrieving a first key from a secure store of a firmware within a platform, the firmware including an initialization table for initializing the platform; and

verifying the initialization table using the first key retrieved from the secure store during an initialization of the platform.

10. The machine-readable medium of claim 9, wherein the initialization table comprises one or more initialization segments that are individually executable, and wherein the method further comprises:

retrieving a second key from the secure store; and

verifying at least one initialization segment using the second key retrieved from the secure store.

11. The machine-readable medium of claim 10, wherein the method further comprises examining the at least one initialization segment to determine whether the at least one initialization segment includes code to be dispatched, wherein the verification is

performed only if the at least one initialization segment includes the code to be dispatched.

12. The machine-readable medium of claim 11, further comprising:
determining whether the verification is performed successfully; and
executing the code dispatched from the at least one initialization segment if the verification is performed successfully.

13. A data processing system, comprising:
a processor;
a memory coupled to the processor to store an initialization table for initializing the data processing system, the memory including a secure store; and
a process, when executed from the memory, causes the processor to
retrieve a first key from the secure store, and
verify the initialization table using the first key retrieved from the secure store during an initialization of the data processing system.

14. The data processing system of claim 13, wherein the initialization table comprises one or more initialization segments that are individually executable, and wherein the process further causes the processor to:
retrieve a second key from the secure store; and
verify at least one initialization segment using the second key retrieved from the secure store.

15. The data processing system of claim 14, wherein the process further causes the processor to examine the at least one initialization segment to determine whether the at least one initialization segment includes code to be dispatched, wherein the verification is performed only if the at least one initialization segment includes the code to be dispatched.

16. A method, comprising:

generating a first key to sign an initialization table of a firmware in a platform, the initialization table being used to initialize the platform;

signing the initialization table using the first key;

storing the first key in a secure store of the firmware; and

locking the secure store after the first key is stored in the secure store.

17. The method of claim 16, wherein the initialization table comprises one or more initialization segments that are individually executable, and wherein the method further comprises:

generating a second key;

signing at least one initialization segment of the initialization table using the second key; and

storing the second key in the secure store prior to locking the secure store.

18. The method of claim 17, further comprising examining the at least one initialization segment to determine whether the at least one initialization segment includes code to be dispatched, wherein the signing operations using the second key is performed only if the at least one initialization segment includes the code to be dispatched.

19. The method of claim 16, wherein the first and second keys are generated, and the initialization table and the at least one initialization segment are signed during a boot process of the platform.

20. A machine-readable medium having executable code to cause a machine to perform a method, the method comprising:

generating a first key to sign an initialization table of a firmware in a platform, the initialization table being used to initialize the platform;

signing the initialization table using the first key;

storing the first key in a secure store of the firmware; and

locking the secure store after the first key is stored in the secure store.

21. The machine-readable medium of claim 20, wherein the initialization table comprises one or more initialization segments that are individually executable, and wherein the method further comprises:

generating a second key;

signing at least one initialization segment of the initialization table using the second key; and

storing the second key in the secure store prior to locking the secure store.

22. The machine-readable medium of claim 21, wherein the method further comprises examining the at least one initialization segment to determine whether the at least one initialization segment includes code to be dispatched, wherein the signing operation using the second key is performed only if the at least one initialization segment includes the code to be dispatched.

23. A data processing system, comprising:

a processor;

a memory coupled to the processor to store an initialization table for initializing the data processing system, the memory including a secure store; and

a process, when executed from the memory, causes the processor to

generate a first key to sign the initialization table,

sign the initialization table using the first key,

store the first key in the secure store, and

lock the secure store after the first key is stored in the secure store.

24. The data processing system of claim 23, wherein the initialization table comprises one or more initialization segments that are individually executable, and wherein the process further causes the processor to:

generate a second key,

sign at least one initialization segment of the initialization table using the second key, and

store the second key in the secure store prior to locking the secure store.

25. The data processing system of claim 24, wherein the process further causes the processor to examine the at least one initialization segment to determine whether the at least one initialization segment includes code to be dispatched, wherein the signing operation using the second key is performed only if the at least one initialization segment includes the code to be dispatched.

26. An apparatus, comprising:

an initialization engine to carry out operations of an initialization table for initializing a platform;

a secure store to perform one or more keys for signing at least a portion of the initialization table; and

a verifier coupled to the initialization engine and the secure store to verify at least a portion of the initialization table using at least one of the one or more keys during an initialization of the platform.

27. The apparatus of claim 26, wherein the one or more keys include a first key, and wherein the initialization engine generates the first key and signs at least a portion of the initialization table using the first key during a boot time of the platform.

28. The apparatus of claim 26, wherein the initialization table includes one or more initialization segments that are individually executable, wherein the one or more keys

include one or more second keys generated for signing the one or more initialization segments respectively during a boot time of the platform.

29. The apparatus of claim 26, wherein during a resume time of the platform, the verifier retrieves the first key from the secure store and verifies the initialization table using the first key.

30. The apparatus of claim 29, wherein for each of the initialization segments that have been signed, the verifier further retrieves the second keys and verifies the signed initialization segments using the second keys respectively.

ABSTRACT OF THE DISCLOSURE

Method and apparatus for protecting a firmware runtime environment are described herein. In one embodiment, a process example is provided to retrieve a first key from a secure store of a firmware within a platform, the firmware including an initialization table for initializing the platform, and verify the initialization table using the first key retrieved from the secure store during an initialization of the platform. Other methods and apparatuses are also described.

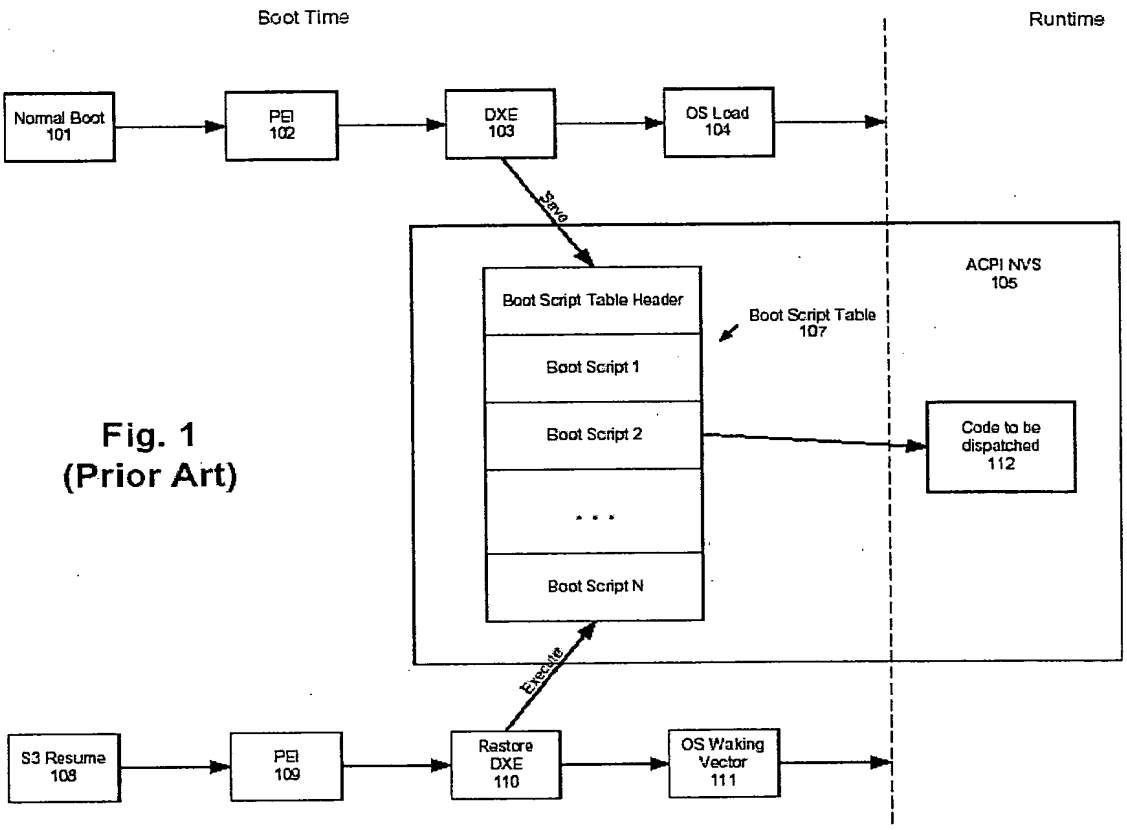
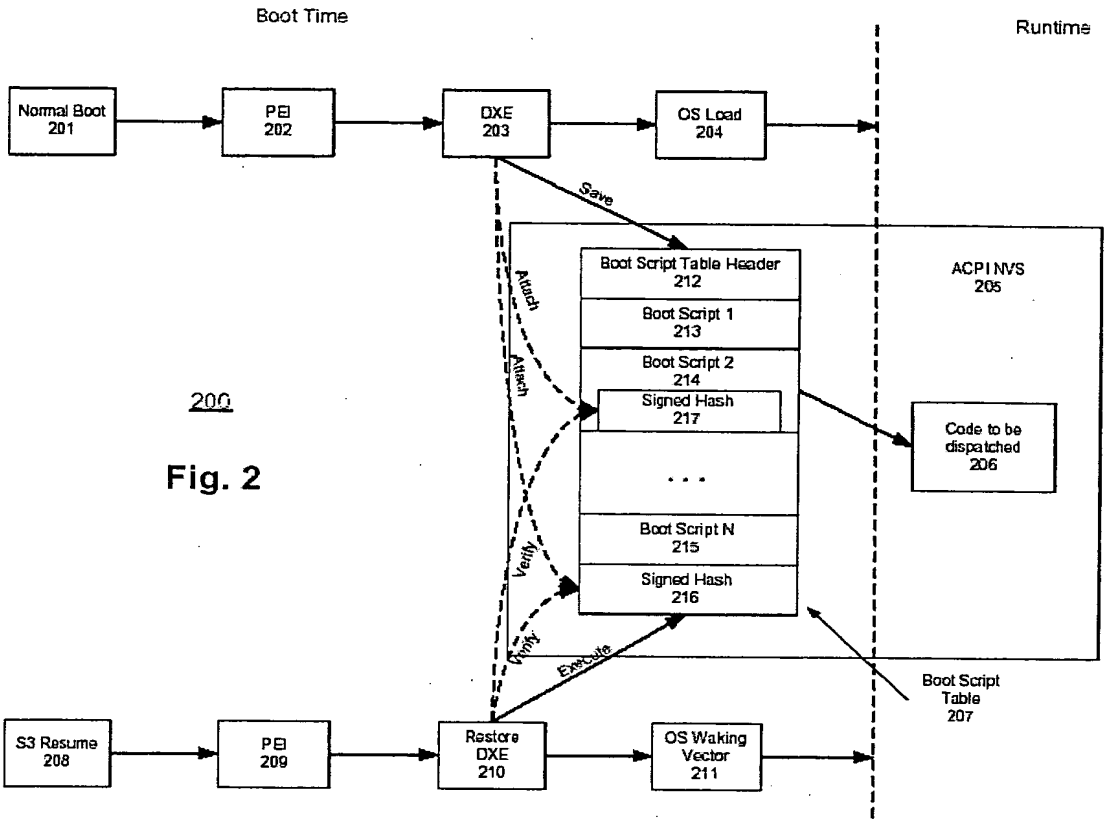


Fig. 1
(Prior Art)


Boot Time
Runtime
Normal Boot 201
PEI 202
DXE 203
OS Load 204
Save
Boot Script Table Header 212
Boot Script 1 213
Boot Script 2 214
Signed Hash 217
. . .
Boot Script N 215
Signed Hash 216
ACPI NVS 205
Code to be dispatched 206
Attach
Attach
Verify
Apply
Execute
Boot Script Table 207
S3 Resume 208
PEI 209
Restore DXE 210
OS Waking Vector 211

200

Fig. 2

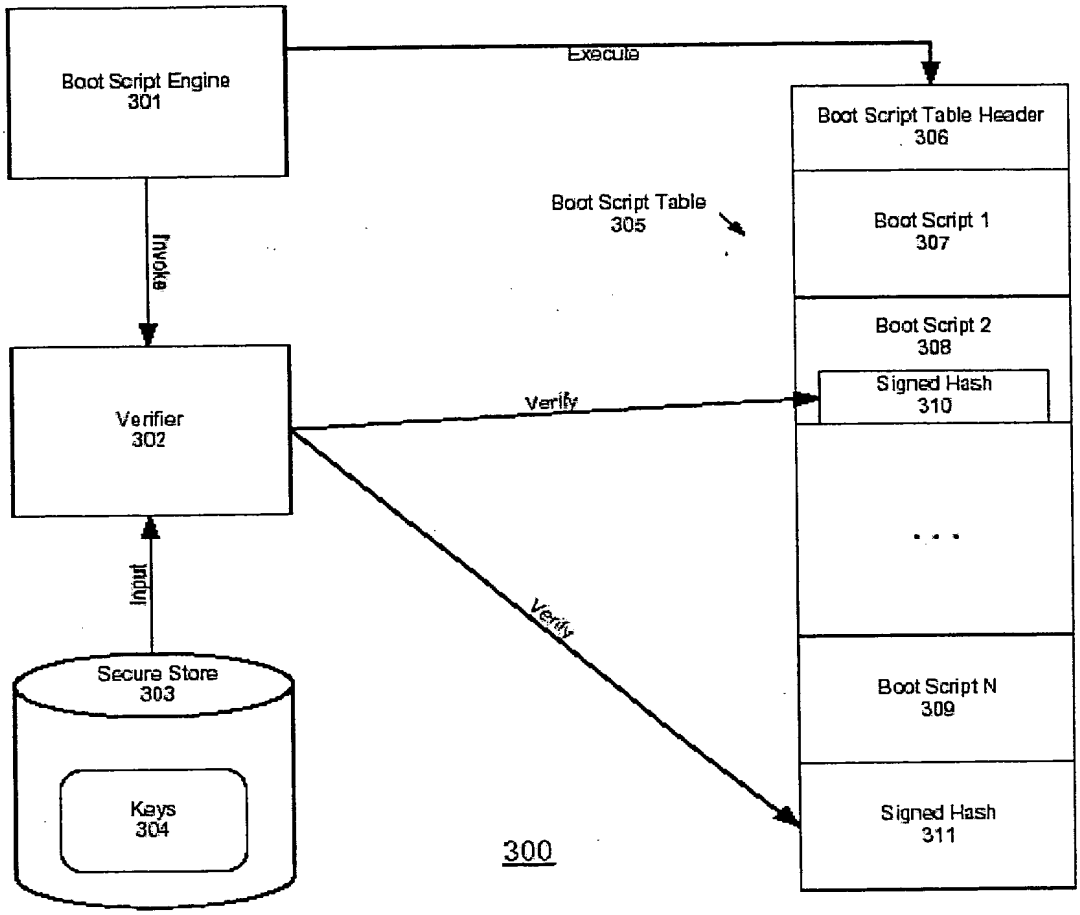
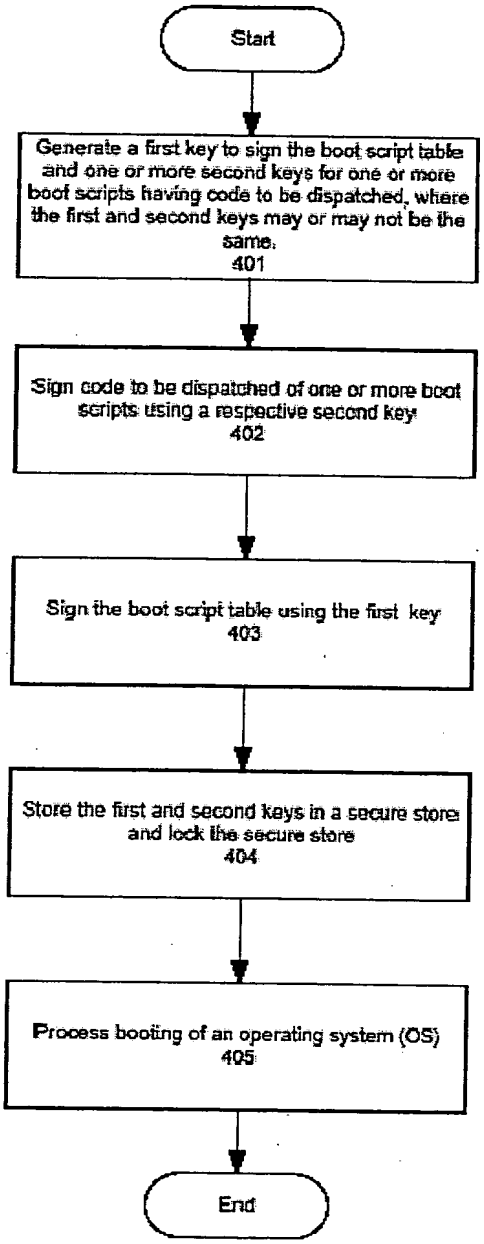

Boot Script Engine
301
Execute
Boot Script Table Header
306
Boot Script Table
305
Boot Script 1
307
Invoke
Boot Script 2
308
Signed Hash
310
Verifier
302
Verify
. . .
Input
Verify
Secure Store
303
Boot Script N
309
Keys
304
Signed Hash
311
300

Fig. 3


Start
Generate a first key to sign the boot script table and one or more second keys for one or more boot scripts having code to be dispatched, where the first and second keys may or may not be the same.
401
Sign code to be dispatched of one or more boot scripts using a respective second key
402
Sign the boot script table using the first key
403
Store the first and second keys in a secure store and lock the secure store
404
Process booting of an operating system (OS)
405
End

400

Fig. 4

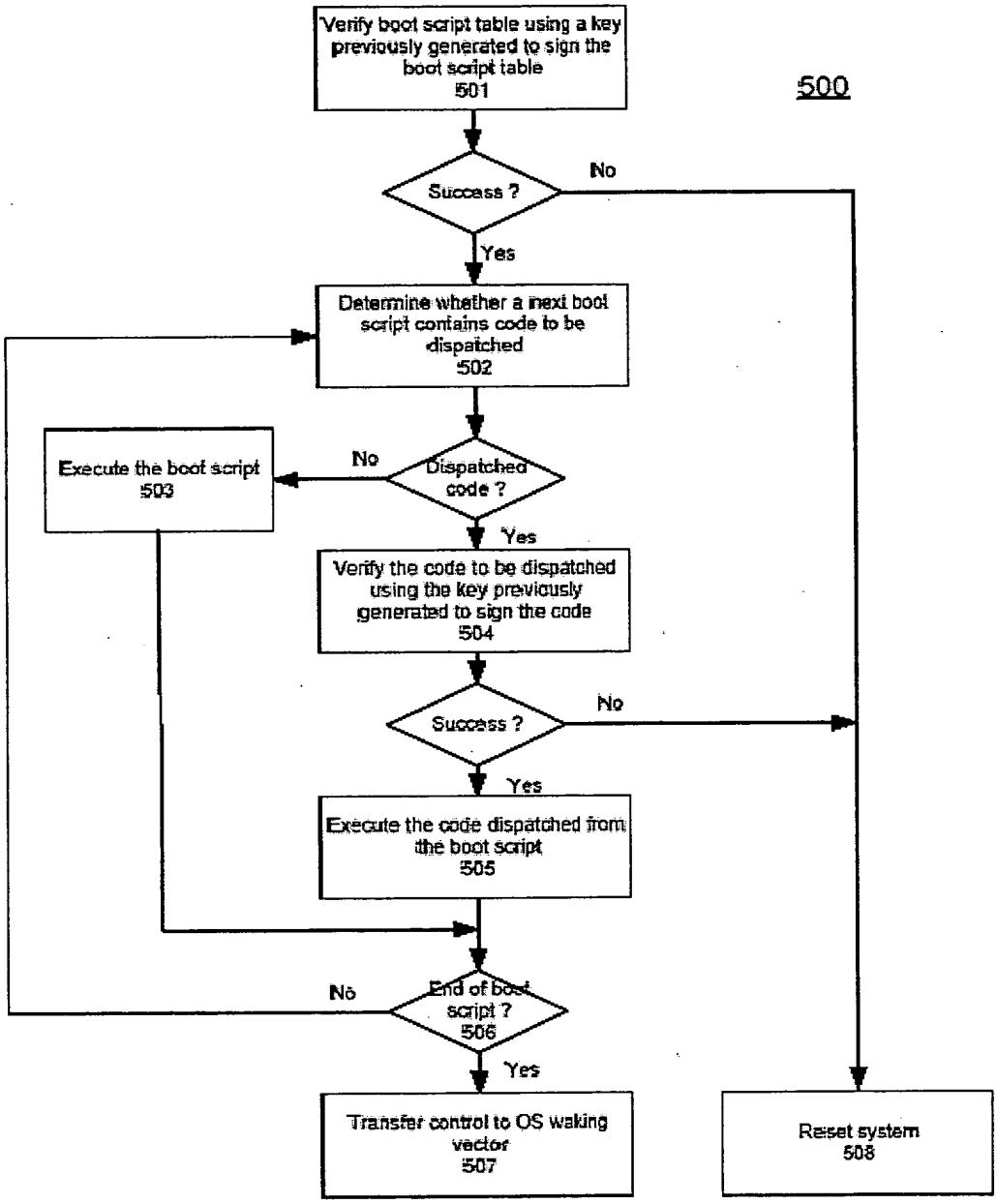

Verify boot script table using a key previously generated to sign the boot script table
501
500
Success ?
No
Yes
Determine whether a next boot script contains code to be dispatched
502
Execute the boot script
503
No
Dispatched code ?
Yes
Verify the code to be dispatched using the key previously generated to sign the code
504
Success ?
No
Yes
Execute the code dispatched from the boot script
505
No
End of boot script ?
506
Yes
Transfer control to OS waking vector
507
Reset system
508

Fig. 5

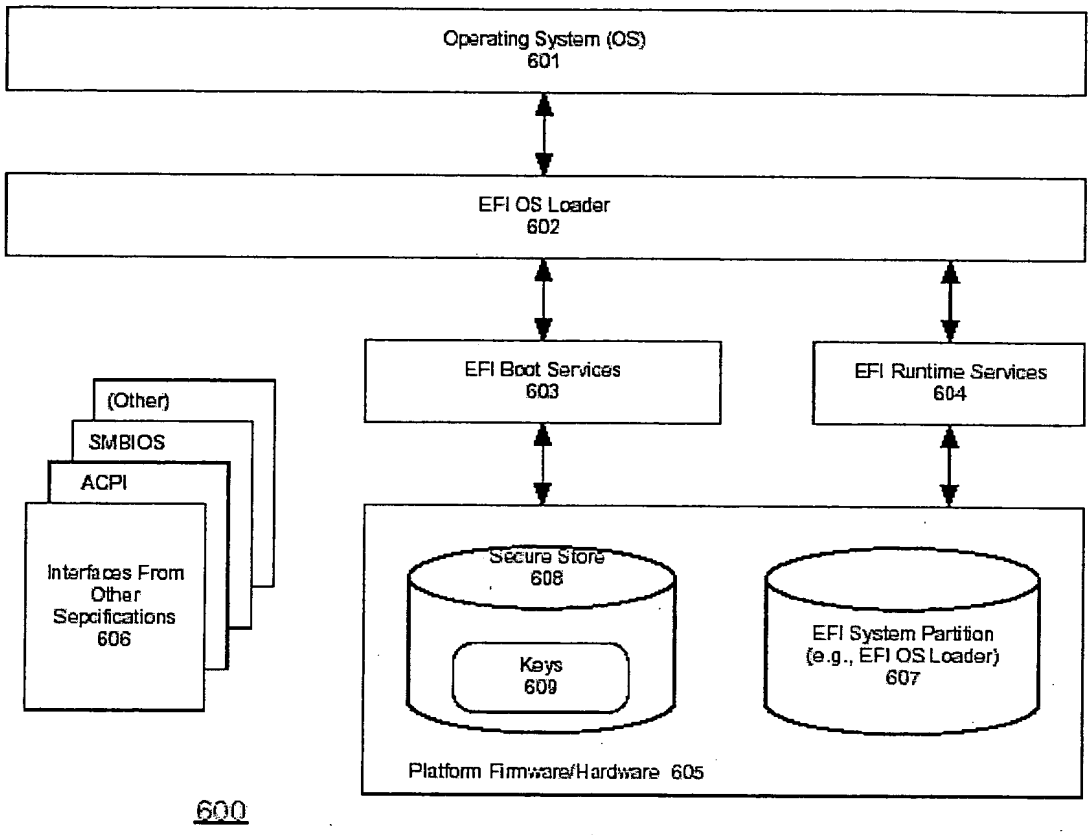

Operating System (OS)
601
EFI OS Loader
602
EFI Boot Services
603
EFI Runtime Services
604
(Other)
SMBIOS
ACPI
Interfaces From
Other
Sepcifications
606
Secure Store
608
Keys
609
EFI System Partition
(e.g., EFI OS Loader)
607
Platform Firmware/Hardware 605
600

Fig. 6

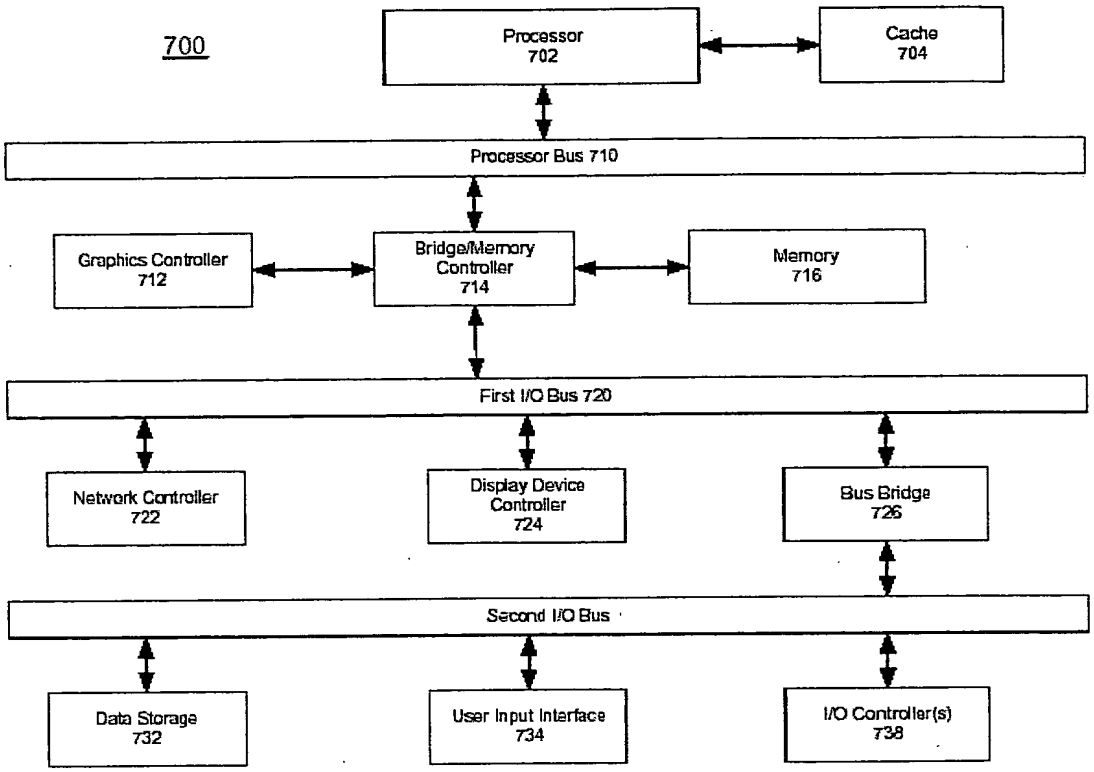

700
Processor
702
Cache
704
Processor Bus 710
Graphics Controller
712
Bridge/Memory
Controller
714
Memory
716
First I/O Bus 720
Network Controller
722
Display Device
Controller
724
Bus Bridge
726
Second I/O Bus
Data Storage
732
User Input Interface
734
I/O Controller(s)
738

Fig. 7